Engineering Optimization
Vol. 00, No. 00, December 2005, 1–22

Hybrid Particle Swarm Optimizer for a Class of Dynamic Fitness Landscape

Susana C. Esquivel[1] and Carlos A. Coello Coello[2]
[1]Laboratorio de Investigación y Desarrollo
en Inteligencia Computacional
Universidad Nacional de San Luis
Ejército de los Andes 950
(5700) San Luis, ARGENTINA
`esquivel@unsl.edu.ar`
[2]CINVESTAV-IPN
Evolutionary Computation Group
Electrical Engineering Department
Computer Science Section
Av. IPN No. 2508, Col. San Pedro Zacatenco
México D.F. 07300, MÉXICO
`ccoello@cs.cinvestav.mx`

(February 2004)

This paper presents the use of Particle Swarm Optimization (PSO) for a class of non-stationary environments. The dynamic problems studied in this work are restricted to one of the possible types of changes that can be produced over the fitness landscape. A hybrid PSO approach (called HPSO_dyn) is proposed, which uses a dynamic macromutation operator whose aim is to maintain diversity. In order to validate the approach, a test case generator previously proposed in the specialized literature was adopted. Such a test case generator allows the creation of different types of dynamic environments with a varying degree of complexity. The main goals of this research were to analyze the ability of HPSO_dyn to react to the changes in the environment, to study the influence of the dynamic macromutation operator on the algorithm's performance and finally, to analyze the algorithm's behavior in the presence of high multimodality.

Keywords: particle swarm optimization, dynamic optimization, non-stationary environments, evolutionary algorithms.

1 Introduction

Many real-world optimization problems are non-stationary. Such problems arise when either the resources (constraints) or the optimization criteria change (or both), and they are common in engineering, production planning, and economics (Michalewicz and Fogel 2000). In the last decade, heuristics that can adapt to changes have attracted the interest of researchers, particularly those that operate on a population of solutions. From these heuristics, most of the research has concentrated on evolutionary algorithms (Angeline 1997, Branke 2002), and several promising results have been obtained within the last few years.

It is worth noticing that several metaheuristics based on swarm intelligence have been adopted in the last few years to solve dynamic optimization problems. Particularly, the use of ant colony optimization (ACO) has increasingly generated interest among researchers (see for example (Bäck et al. 1997, Caro and Dorigo 1998, Schoonderwoerd et al. 1996, Guntsch and Middendorf 2001, Guntsch et al. 2001, Guntsch and Middendorf 2002*a*,*b*)).

Another technique that has recently been adopted for dealing with non-stationary environments is Particle Swarm Optimization (PSO) (Kennedy and Eberhart 2001, Carlisle 2002), which is precisely the approach adopted in the work reported in this paper.

The remainder of the paper is organized as follows. Section 2 provides a short introduction to the so-called "classical" particle swarm optimization model. Section 3 reviews the previous related work. Section 4 describes the proposed approach. Since the test case generator (of dynamic test functions) proposed by Morrison & De Jong (1999) was adopted, in Section 5 a brief description of such generator is given in order to make this paper self-contained. The experimental design adopted for the study is described in Section 6. The metrics adopted to evaluate the approach performance are described in Section 7. The analysis of results is presented in Section 8. Finally, the conclusions and some possible paths for future research are provided in Section 9.

2 Particle Swarm Optimization

The PSO model (now considered "classical") was originally proposed by James Kennedy and Russell Eberhart (Kennedy and Eberhart 1995), and has had several further extensions in the next few years (Kennedy and Eberhart 2001). The PSO model is inspired on the acting of communities that present both an individual and a social behavior, from which the main socio-cognitive ideas may be applied to the development of efficient algorithms that solve optimization problems (this is the so-called "swarm intelligence" (Kennedy and Eberhart 2001)). Although PSO is considered by its authors as an evolutionary algorithm, it does not incorporate the traditional genetic operators (crossover and mutation) that evolutionary algorithms normally adopt. Instead, in PSO each particle adjusts its flight over the search space based on its own flight experience and that of its neighbors (i.e., the other particles of its swarm).

Each particle represents a possible solution to the problem at hand and is treated as a point in an n-dimensional search space (where n refers to the number of decision variables of the problem). A particle is characterized by its position and its current velocity. Furthermore, a particle has a memory where it stores the best position that it has found so far. Particles adjust their flight trajectories using the following equations (Shi and Eberhart 1998):

$$v_{i,j} = w \times v_{i,j} + c_1 \times r_1 \times (p_{i,j} - x_{i,j}) + c_2 \times r_2 \times (p_{g,j} - x_{i,j}) \tag{1}$$

$$x_{i,j} = x_{i,j} + v_{i,j} \tag{2}$$

where w is the inertia factor, whose goal is to control the impact of the history of the velocities of a particle over the current velocity, by influencing the local and global exploration abilities of the algorithm. $v_{i,j}$ is the velocity of the particle i in the j-th dimension, c_1 and c_2 are weights applied to the influence of the best position found so far by particle i and by the best particle in the swarm g. r_1 and r_2 are random values (with a uniform distribution) generated within the interval $[0, 1]$. After the velocity is updated, the new position of the particle i in its jth dimension is recomputed. This process is repeated for each dimension of the particle i and for all the particles in the swarm. Kennedy and Mendes (2002) proposed different neighborhood techniques to be considered when updating the velocity of a particle. In this paper, the most simple neighborhood scheme available was adopted, which only includes the closest neighbors in terms of the indices of the particles. Under this scheme, the equation to update the velocity of a particle becomes:

$$v_{i,j} = w \times v_{i,j} + c_1 \times r_1 \times (p_{i,j} - x_{i,j}) + c_2 \times r_2 \times (p_{l,j} - x_{i,j}) \tag{3}$$

where $p_{l,j}$ represents the best particle in the neighborhood l. The formula used to update a particle remains the same as before.

Despite the popularity of PSO as an (static) optimizer (see for example (Kennedy and Eberhart 2001, van den Bergh 2002, Ray and Liew 2002, He et al. 2004, Fan et al. 2004)), its use in dynamic environments has been fairly limited, as we will see in the next section.

3 Previous Work

An algorithm capable of dealing with non-stationary environments normally considers two main aspects (Branke 2002): the detection of the change and the reaction of the algorithm to such change, such that the (moving) optimum can be tracked as closely as possible. There is some previous research on the use of particle swarm optimization for dynamic optimization that involves these two aspects previously discussed. Such previous work will be briefly reviewed in this section.

Carlisle & Dozier (2000) used a *sentry particle* that is evaluated at each iteration, comparing the previous fitness value with the new one (if they differ, this means that the environment has changed). Hu & Eberhart (2002) proposed the re-evaluation of two g_{best} particles in order to detect the movement of the location of the optimum.

In further work, Carlisle & Dozier (2002) suggested periodic re-evaluation of the personal best and the swarm fitness values, but only when a change has occurred. Janson and Middendorf (2004) proposed a Hierarchical PSO (PH-PSO) in which, whenever a change is detected by the algorithm, the rank (or hierarchy) is split in a number of sub-swarms. The detection of a change is produced by re-evaluating the position of the best particle (g_best) at each flight cycle. If the function value at such position has changes with respect to the previous value, then a change has occurred and the algorithm starts its reactive stage which consists of re-randomizing a portion of the swarm and re-evaluating the rest.

Blackwell and Branke (2004) extended both the basic (single population) PSO and the so-called CPSO (*Charged Particle Swarm Optimization*, which was proposed by the same authors (Blackwell 2003)) by building interactive multi-swarms. The underlying idea is that the charged multi-swarms maintain diversity in the population, which allows that the changes can be automatically detected and also allows the swarm to adapt to such changes.

Parrot & Li (2004) use PSO to tackle peaks of a function in dynamic environments. They also adopt the dynamic functions generator proposed by Morrison & De Jong (1999). In their paper, they compare four approaches: classical PSO, FGPSO (Fine-grained PSO), PSO-R20 and PSO-R50. Although not clearly stated in the paper, both PSO-R20 and PSO-R50 use a macro-mutation operator taken from the evolutionary computation literature. They adopt the strategy of Random Immigrants which replaces a portion of the population with randomly generated particles. PSO-R20 replaces 20% of the population, whereas PSO-R50 replaces 50% of the population. FGPSO works with a neighborhood, using a grid topology in which each particle has four neighbors: located up, down, right and left. In this work, the authors used a single measure of performance (the distance between the best particle found in each interval within changes and the value of the optimum particle). Note however, that the use of only this measure of performance may be misleading in some cases since this measure may provide good values even if the approach never reaches the optimum peaks.

Despite the encouraging results reported in the papers previously indicated, most of these authors focus their work on unimodal fitness landscapes (Blackwell 2003, Carlisle and Dozier 2002) which is rather unrealistic considering the complexity of real-world problems, which tend to present multimodality. This is precisely the issue addressed in this paper. Aiming to provide a framework for performing further comparative studies, the test case generator originally developed by Morrison & De Jong (1999) was adopted, which shares several similarities with the proposal of Branke (2002). This test case generator allows to define non-stationary environments of different degrees of complexity both regarding the morphology of the fitness landscapes as well as regarding the type of changes that can be produced and their severity.

4 The Proposed Approach

The proposal presented in this paper uses the equations (2) and (3) to update the particle positions and its velocities, respectively. The local PSO model was selected because in a previous work (Esquivel and Coello Coello 2003), it was shown that this model works better than the global PSO model for functions with high multimodality.

When a PSO algorithm is executing and it converges to an optimum, the swarm loses the diversity necessary for exploring the search space and consequently, it loses its ability to react to a change when such a change occurs.

Together with the proposal to maintain diversity at every interval between changes, a dynamic mutation operator was also introduced. This operator is initialized with a high mutation probability ($pmax$) at the beginning of each interval, and then it is decreased over the interval until reaching its lowest allowable value ($pmin$). This operation causes a more explorative behavior at the beginning of the search and a more focused behavior (i.e., narrowing the search to a much smaller region of the search space) at the end, right before a change occurs. The dynamic mutation operator is described next.

Mutation: Each coordinate of the particle is independently mutated with a probability p_{mut}. The coordinate value is replaced by another value which is randomly generated within its allowable range. The mutation probability is dynamically adjusted taking values within the interval $[p_{min}, p_{max}]$ during the execution of the algorithm. Thus, p_{mut} is defined as follows:

```
1. Initialize($Swarm$)
2. Initialize($velocities$)
3. Evaluate($Swarm, F_0$)
4. Copy($Swarm, Swarm\_bests$)
5. $t = 0$
6. do
7.     Calculate($p_{mut}$)
8.     if ($occurred\_change$)
9.         Report_dynamic_statistics()
10.        Change_function()
11.        Evaluate($Swarm, F_t$)
12.        Evaluate($Swarm\_bests, F_t$)
13.        Update($Swarm\_bests$) if appropriate
14.        Calculate($p_{mut}$)
15.    end if
16.    Mutate($Swarm$)
17.    Update($velocities$)
18.    Update($Swarm$)
19.    Evaluate($Swarm, F_t$)
20.    Update($Swarm\_bests$) if appropriate
21.    $t = t + 1$
22. while ($\neg termination$)
```

Figure 1. General outline of our HPSO_dyn Algorithm

$$p_{mut} = \frac{(p_{max} - p_{min}) \times (interval - iter_{current} \bmod interval)}{interval} + p_{min} \tag{4}$$

where the values p_{min} and p_{max} are the lower and upper bounds of the variation interval of the probability, $interval$ indicates the number of iterations between changes and $iter_{current}$ corresponds to the current flight cycle of the particles. Each time a change takes place in the environment, the value of p_{mut} is initialized to p_{max} and the end of the interval of p_{mut} is set to p_{min}.

The pseudo-code of the approach (called Hybrid PSO for dynamic optimization, or HPSO_dyn, for short) is shown in Figure 1.

Once the swarm and the velocities are initialized (lines 1–2), the swarm is evaluated with the base function F_0. Then, the memory of the particles ($Swarm\_bests$) is initialized (line 4). After that, the algorithm enters the flight loop (line 6) which is executed while the termination condition (line 22) is true. This condition indicates the total number of iterations, that depends of the number of changes to be performed and the lenght of the interval between such changes (for example, if the number of changes that will take place in a run is 20 and each change is produced every 30 iterations, then the total number of iterations is 20 × 30). Then, in line 7, we compute the value for p_{mut} (according to equation (4)) and the function `occurred_changes` determines if a change is required within the current flight cycle. In order to do this, the function checks if the current cycle number is a multiple of the number of cycles between changes. The changes in the environment are produced at constant intervals. If the environment must change, the dynamical statistics are stored (i.e.,the data necessary to compute the metrics defined in Section 7), the function is modified and both the *Swarm* and the *Swarm_bests* are re-evaluated with the new fitness function (lines 9 to 12). These re-evaluations are necessary since the current fitnesses of the particles do not correspond to the new function. Once the two swarms have been re-evaluated, for each particle is verified that *Swarm_bests* really contains the best particle positions for the new environment, as to maintain the consistency of the algorithm. By doing this, the memory of all the particles is not lost, but only the memory of those for which their current positions (in *Swarm*) are the best (Carlisle and Dozier 2002). Then, the transition function for the probability of mutation is recomputed setting p_{mut} to p_{max}. Next, the algorithm proceeds with the normal PSO processing, which implies: mutate the particles, update their velocities, update the positions of the particles, evaluate the particles and, if applicable, modify their position in *Swarm_bests* (lines 16–20). This process is done asynchronously, since there is evidence of the efficiency of this type of processing when working with neighborhoods in PSO (Esquivel and Coello Coello 2003, Carlisle 2002).

5 DF1 Generator

In this section, the Test Function Generator proposed by Morrison & De Jong (1999) is briefly described. This generator uses a morphology that the authors denominate “field of cones”. Such cones can have different heights and slopes and are randomly distributed along the landscape. The static base function (F_0), for the two-dimensional case, is defined as:

$$F_0(x,y) = \max_{i=1,n}[H_i - R_i \times \sqrt{(x-x_i)^2 + (y-y_i)^2}] \tag{5}$$

where n indicates the number of cones in the landscape and each cone is independently specified by its coordinates (x_i, y_i) that belong to the interval $[-1, 1]$, its height (H_i) and its slope (R_i). The cones independently defined are grouped using the function *max*. Each time the generator is invoked, it randomly generates a morphology with the characteristics given. The function F can be defined for any number of dimensions. The user can create a wide variety of shapes for the fitness landscape by specifying the range of allowable values for the height, slope and location of the cones. Furthermore, the generator allows to re-write the values randomly generated in order to create landscapes with controlled characteristics. The severity of the changes is controlled through the *logistic function*:

$$Y_i = A \times Y_{i-1} \times (1 - Y_{i-1}) \tag{6}$$

where A is a constant defined within the interval $[1, 4]$ and Y_i is the value at the iteration i. Thus, the procedure to obtain the dynamic behavior that the user wants is the following: 1) choose the number of cones and 2) determine what characteristics the user wishes to change (height, location, slope of one or all the cones). The severity of the change is controlled by the values that are assigned to the constant A, which influences the movement produced, either at small, large or chaotic steps. For further details on this generator, the reader should refer to (Morrison and De Jong 1999).

6 Experimental Design

The goal of this research was triple: first, to determine if the proposed algorithm could track down the optimum once a change has occurred. Second, to verify the influence of the dynamic mutation operator on the algorithm's performance and, finally, to analyze the algorithm's behavior upon scaling both the number of dimensions and the number of cones. Thus, this article aims to extend the scenarios proposed by Morrison (2002) in order to achieve the required conditions to perform the study. Such scenarios are described next: 1) by the shapes of the fitness landscape and 2) by the dynamics applied.

Description of the structure of the static landscape

(i) E1: 2 dimensions, 5 cones (2d-5c).
(ii) E2: 2 dimensions, 14 cones (2d-14c).
(iii) E3: 5 dimensions, 5 cones (5d-5c).
(iv) E4: 5 dimensions, 14 cones (5d-14c).
(v) E5: 10 dimensions, 5 cones (10d-5c).
(vi) E6: 10 dimensions, 14 cones (10d-14c).

In all the scenarios, the maximum value was assigned to a single peak. Such

Table 1. DF1 Parameters for Scenarios E1, E3 and E5

Parameter	Small	Large	Chaotic
Hbase	60.0	6.0	60.0
Hrange	0.0	0.0	0.0
Rbase	70.0	70.0	70.0
Rrange	0.0	0.0	0.0
A	1.5	1.5	3.8
Scale	0.3	0.99	0.5
OptH	90.0	90.0	90.0
OptR	90.0	90.0	90.0

Table 2. DF1 Parameters for Scenarios E2, E4 and E6

Parameter	Small	Large	Chaotic
Hbase	1.0	1.0	1.0
Hrange	9.0	9.0	9.0
Rbase	8.0	8.0	8.0
Rrange	12.0	12.0	12.0
A	1.5	1.5	3.8
Scale	0.3	0.99	0.5
OptH	15.0	15.0	15.0
OptR	20.0	20.0	20.0

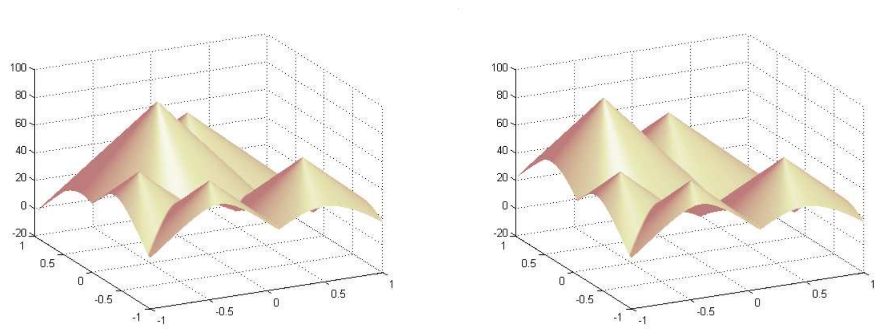
a value is greater than that of the other peaks (to make sure that there is a single global optimum).

Dynamics Applied

The type of change implemented consisted of modifying the location of all the cones simultaneously and in all the scenarios was applied the following severity: small, large, and chaotic. For the scenarios $E1$ to $E4$, the changes take place at every 10, 30 and 50 iterations. For scenarios $E4$ and $E5$, given their complexity, changes take place at every 30, 60 and 90 iterations. Furthermore, at each run, 20 changes took place on the location of all the cones.

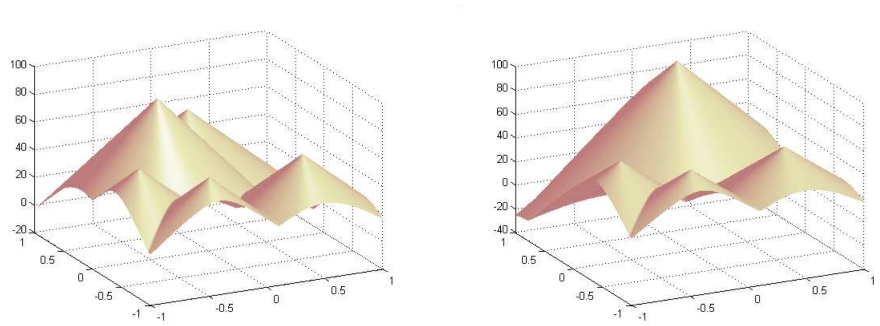
Parameters of the DF1 Generator

The parameters of DF1 that correspond to the dynamics implemented are defined in Tables 1 and 2, according to the proposal by (Morrison 2002), where $Hbase$, $Hrange$, $Rbase$ and $Rrange$ correspond to the ranges of the heights and slopes of the cones that do not contain the global optimum. *OptH* and *OptR* correspond to the cone containing the global optimum. A and $Scale$ control the severity of the changes. In all the experiments, the variable Y in equation (6) is initialized with the value 0.45 (this is the default value provided with the generator).



(a) (b)

Figure 2. (a) Base Function: 2 Dimensions, 5 Cones - (b) Small Change



(a) (b)

Figure 3. (a) Base Function: 2 Dimensions, 5 Cones - (b) Large Change

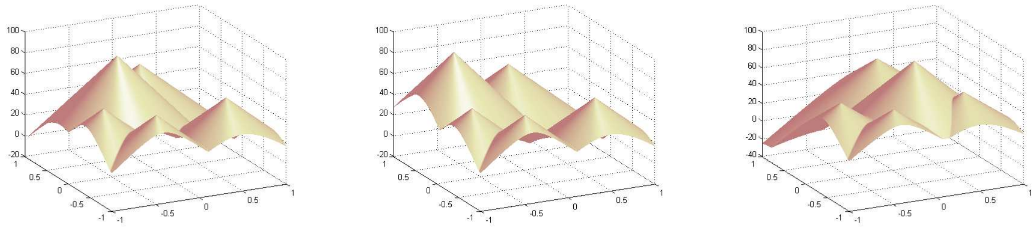
In Figures 2, 3 and 4, examples of the types of static landscapes that can be generated with DF1 when subject to changes with different degrees of severity are shown.

Figure 2 (a), shows a static base function of two dimensions and five cones.

In Figure 2 (b), the landscape modified by a small change can be observed. Figure 3 (b) illustrates the landscape modified by a large change and finally, Figures 4 (b) and 4 (c), display two successive chaotic changes. A similar example but for static base functions of two dimensions and fourteen peaks is shown in Figures 5, 6 and 7.

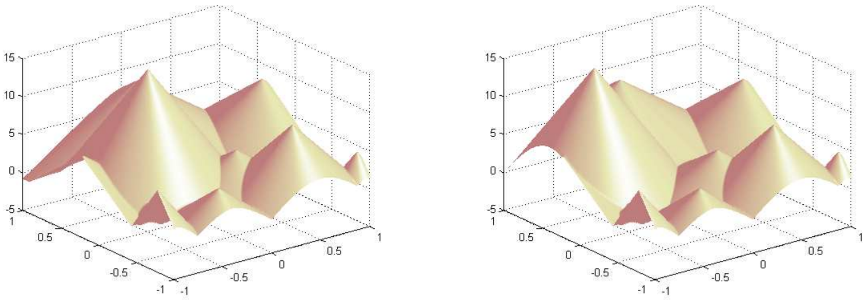
Parameters of the Classical PSO algorithm

In (van den Bergh 2002), the influence of the parameter settings on the



(a) (b) (c)

Figure 4. (a) Base Function: 2 dimensions, 5 Cones - (b) Chaotic Change (1) - (c) Chaotic Change (2)



(a) (b)

Figure 5. (a) Base Function: 2 dimensions, 14 Cones - (b) Small Change

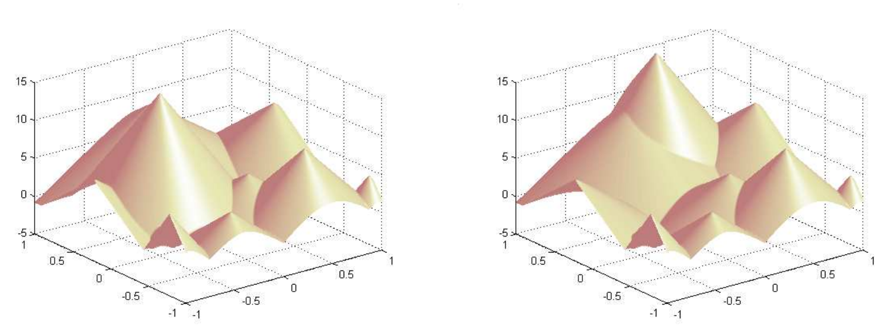
Table 3. Parameter Configuration

Parameters	Values							
w	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3
c_1, c_2	2.0	1.9	1.8	1.7	1.6	1.5	1.4	1.3

performance and convergence of the PSO algorithm were both empirically and theoretically studied. As suggested in this study, in the present work a set of previous experiments were performed in order to select the best combination of values for w, c_1, and c_2 for the problem under consideration. The settings considered are listed in Table 3.

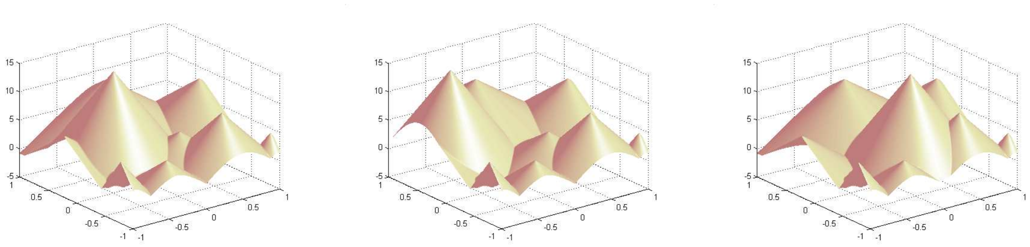
Note that all the configurations hold the convergence relation defined in (van den Bergh 2002):

$$\frac{c_1 + c_2}{2} - 1 \leq w \tag{7}$$



(a) (b)

Figure 6. (a) Base Function: 2 dimensions, 14 Cones - (b) Large Change



(a) (b) (c)

Figure 7. (a) Base Function: 2 dimensions, 14 Cones - (b) Chaotic Change (1) - (c) Chaotic Change (2)

Table 4. PSO Algorithm Parameter Settings

Scenarios	w	$c_1 = c_2$	$swarm_{size}$
E1-E2	0.5	1.5	50
E3-E4	0.5	1.5	200
E5-E6	0.5	1.5	500

This preliminary experimentation showed that the best performance of PSO was obtained with the combination of parameter values listed in Table 4. Analogously, different neighborhood radius were analyzed resulting the best, for this problem class, a neighborhood radius of size 4.

In each experiment, 30 runs were performed, all of them with the same base function and the same initial population. The rationale to work with the same initial population was to verify the robustness of the algorithms with respect to the results obtained.

Parameters of the HPSO_dyn algorithm

Table 5. E1 (2d-5c): Probability interval and changes tracked down

Interval	Small	Large	Chaotic
[0.0, 0.0]	50	30	10
[0.1, 0.2]	510	500	498
[0.1, 0.3]	600	593	583
[0.1, 0.4]	600	600	600

Table 6. E3 (5d-5c): probability interval and changes tracked down

Interval	Small	Large	Chaotic
[0.1, 0.4]	593	583	581
[0.1, 0.5]	597	597	587
[0.1, 0.6]	599	596	588
[0.2, 0.4]	600	599	585
[0.2, 0.5]	594	599	585
[0.2, 0.6]	594	596	590
[0.3, 0.4]	600	597	587
[0.3, 0.5]	600	595	591
[0.3, 0.6]	600	600	600

Table 7. E5 (10d-5c): probability interval and changes tracked down

Interval	Small	Large	Chaotic
[0.3, 0.6]	390	294	299
[0.4, 0.7]	475	448	424
[0.5, 0.8]	600	596	542

As indicated in Section 4, the classical PSO algorithm was hybridized with a dynamic macromutation operator. A set of preliminary experiments were performed with two purposes: a) to determine which were the most appropriate values for the p_{min} and p_{max} macromutation operator parameters, and b) to study the way in which the modification of these values influenced the algorithm's reaction to changes in the environment. These preliminary experiments were performed considering the most unfavorable cases, that is, when the changes are produced at every 10 iterations for scenarios $E1$ and $E3$, and at every 30 iterations for scenario $E5$. Only the five peaks functions were taken into account because in some previous work of the authors (Esquivel and Coello Coello 2004), it was found that the algorithm had more difficulties when increasing dimensionality rather than when increasing multimodality.

The results are presented in Tables 5 to 7.

As can be observed in Table 5, the HPSO_dyn algorithm was able to react only to a very limited number of changes when the macromutation operator was not adopted.

The results from Table 6 show that for changes produced with small and large severity, the values assigned to p_{min} and p_{max} do not significantly increase the number of changes at which the algorithm properly reacts. However, it

Table 8. Definitive Probability Intervals

Scenarios	Interval
E1-E2	$[0.1, 0.4]$
E3-E4	$[0.3, 0.6]$
E5-E6	$[0.5, 0.8]$

can be observed, in the case of changes produced with chaotic severity, that when p_{min} and p_{max} are increased, the algorithm reacts better to the changes. This observation allows to infer that the algorithm requires a high diversity in the swarm not only at the first iterations, but also during the full interval, including the iteration just before the change occurs.

In scenario $E5$, considering the complexity of the landscape studied, the results obtained are considered very satisfactory for changes of small and large severity. The behavior is also considered satisfactory for the chaotic case, since the algorithm was not able to react properly in less than 10% of the changes performed.

The minimum and maximum values for the macromutation operator, for each scenario, are summarized in Table 8.

7 Performance Metrics

The performance of the HPSO_dyn was evaluated using three performance metrics: the first one considers the quality of the solutions obtained, the second one computes the error between the best fitness value found by the algorithm and the optimum value (the test function generator facilitates this value) and the last one indicates the number of changes to which the algorithm reacts appropriately. The description of these 3 performance metrics is provided next.

Average Mean Best Fitness (AMBF): This metric provides the average mean value of the best particle values at the iteration "just before a change occurs". It is defined as:

$$AMBF = \frac{1}{rn} \sum_{j=1}^{rn} \left(\frac{1}{cn} \sum_{i=1}^{cn} fbest_{ij} \right) \tag{8}$$

where rn is the number of runs, cn is the number of changes performed in one run, and $fbest_{ij}$ is the fitness value of the best particle found at the i-th change at the j-th run.

Average Mean Error (AME): In this case, the metric gives the average of

Table 9. AC Metric for 2D Functions

	E1: 2d - 5c			E2: 2d - 14c		
Interval	Small	Large	Chaotic	Small	Large	Chaotic
10	600	600	600	600	600	600
30	600	600	600	600	600	600
50	600	600	600	600	600	600

Table 10. AC Metric for 5D Functions

	E3: 5d - 5c			E4: 5d - 14c		
Interval	Small	Large	Chaotic	Small	Large	Chaotic
10	600	600	600	588	581	575
30	600	600	600	600	598	599
50	600	600	600	600	600	600

the mean error value, i.e., considering the error as the difference between the optimum value and the best value found by the algorithm at the iteration "just before a change occurs". It is defined as:

$$AME = \frac{1}{rn}\sum_{j=1}^{rn}(\frac{1}{cn}\sum_{i=1}^{cn}(opt - fbest_{ij})) \qquad (9)$$

where *opt* is the optimum value.

Amount of Changes (AC): To be sure that our HPSO_dyn successfully tracks down the cone that contains the optimum value when it is moving on the landscape and to ensure that there is no misinterpretation of the results, for each cone in any function, the test generator DF1 allows to know its location, height and slope. Thus, it is possible to know if the best particle belongs to the optimum peak. Therefore, this metric computes the number of times in which the best particle was located on the optimum cone. In one experiment the algorithm can detect a maximum of 600 changes (20 changes per run times 30 runs).

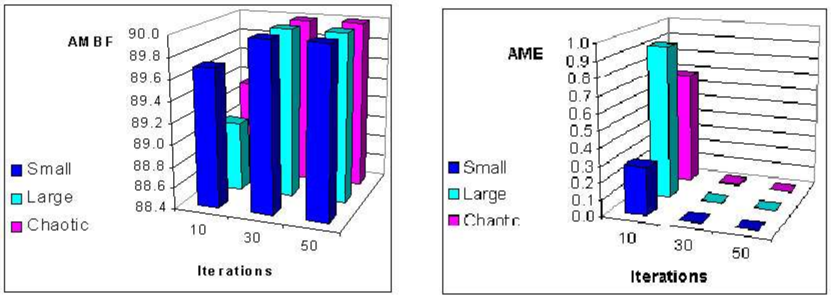
8 Analysis of Results

The optimum value for all the functions with 5 cones is 90.00 and all the other cones have a maximum value of 60.00. For the functions with 14 cones, the maximum value is 15.00 and all the other cones have a maximum of 10.00.

The performance of HPSO_dyn for the two-dimensional functions with 5 and 14 cones is very satisfactory, since in all cases, for all the changes performed, the algorithm was able to react to the 100% of the changes (see Table 9). Also, the algorithm was able to track down the global optimum with a very acceptable average mean fitness (see Figures 8 (a) and 9 (a)) and average mean

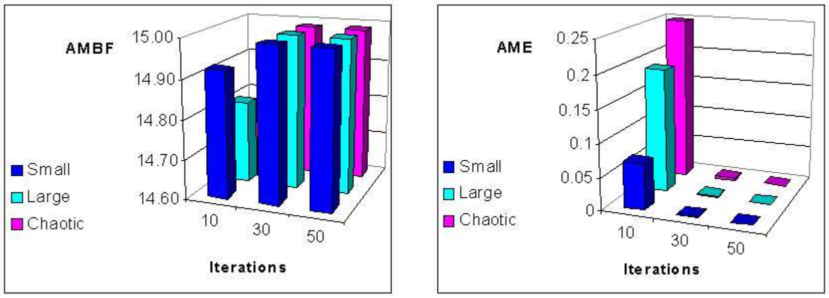
Table 11. AC Metric for 10D Functions

	E1: 10d - 5c			E2: 10d - 14c		
Interval	Small	Large	Chaotic	Small	Large	Chaotic
30	600	596	542	600	600	527
60	600	600	598	600	599	550
90	600	600	600	600	599	560



(a) (b)

Figure 8. (a) 2d-5c AMBF (b) 2d-5c AME



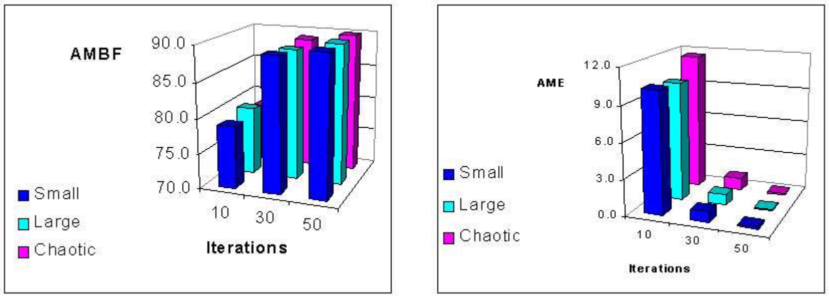
(a) (b)

Figure 9. (a) 2d-14c AMBF (b) 2d-14c AME

error values (see Figures 8 (b) and 9 (b)).

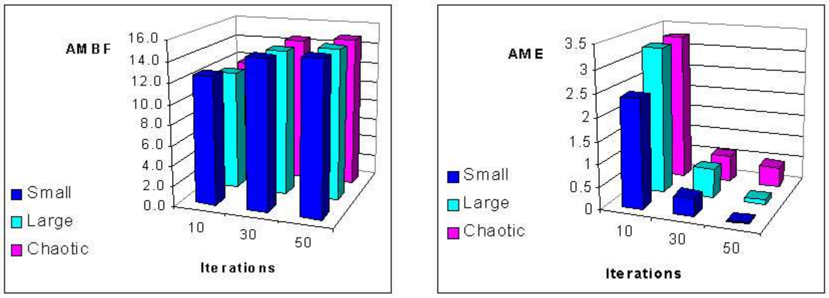
For the case of the 5-dimensional functions with 5 and 14 cones the performance of HPSO_dyn remains satisfactory regarding the number of changes to which the algorithm reacts correctly. Table 10 shows that, even for the 14 peaks function when chaotic changes are performed at every 10 iterations, in the worst case the algorithm reacts to 96% of the changes produced.

With respect to the metric AMBF (see Figures 10 (a) and 11 (a)), and AME (see Figures 10 (b) and 11 (b)), an increase in the error can be observed, particularly when the changes are produced at every 10 iterations. This indicates that even though the algorithm properly reacts to the changes, the values that



(a) (b)

Figure 10. (a) 5d-5c AMBF (b) 5d-5c AME



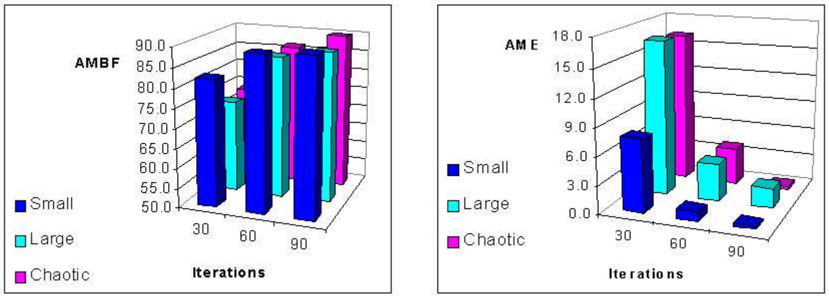
(a) (b)

Figure 11. (a) 5d-14c AMBF (b) 5d-14c AME

it finds are more distant from the optimum value.

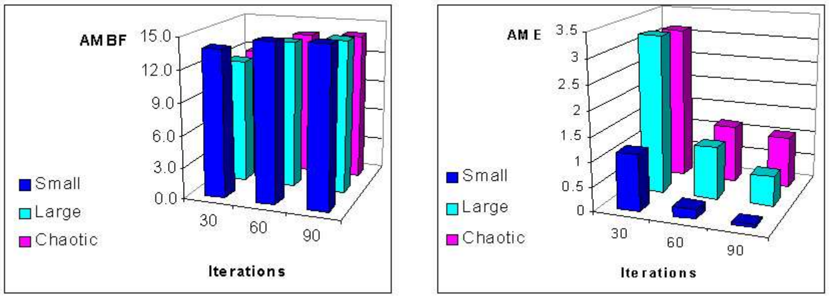
For the more complex scenarios (10 dimensional functions), when the changes are produced with small and large severities, the algorithm's performance was good if we consider the AC metric (see Table 11). For changes produced with chaotic severity, the algorithm's performance degraded, since for the 14 cones case it could not succeed 100% of the time at tracking down the optimun in any of the experiments. Nevertheless, the success rate oscillated between 87.8% and 99.6%, which are reasonably good values.

It is also worth noticing the increase in values of the metrics AMBF and AME, which is shown in Figures 12 and 13. This increase is produced as a result of the ocurrence of more unsuccessful cases.



(a) (b)

Figure 12. (a) 10d-5c AMBF (b) 10d-5c AME



(a) (b)

Figure 13. (a) 10d-14c AMBF (b) 10d-14c AME

9 Conclusions and Future Work

This paper presented an optimization algorithm for non-stationary environments. The algorithm is based on a particle swarm optimizer which was hybridized with a dynamic macromutation operator. Although one could think at first sight that the mutation probabilities adopted in the work are too high, this is done to maintain the required diversity in the swarm. However this operator does not disrupt the search process since the memory *Swarm_bests* is not destroyed when the changes take place. Additionally *Swarm_bests* is only updated when a particle from the *Swarm* obtains a better fitness than the one stored in such memory.

Furthermore, some empirical evidence was provided to validate the hypothesis that the algorithm without the dynamic macro-mutation operator adopted would be able to react only to a fairly limited number of changes. This indicates that the macro-mutation operator fulfilled its design goal, which was to maintain the required diversity within the swarm.

In the experiments, functions with 2, 5 and 10 dimensions and with 5 and 14 cones were involved. All of these functions were created with the DF1 generator, which allowed to study the behavior of the algorithm with more complex fitness landscape structures than those normally adopted in the specialized literature (i.e., the sphere model). The changes produced consisted on changing the location of all the cones, with step sizes that were from small to chaotic. The results obtained show that the performance of the algorithm over morphologies with several suboptima, is highly satisfactory when using either 2 or 5 dimensions and a number of cones between 5 and 14. However, the performance degrades as we move to problems with 10 dimensions. Nevertheless, the success rate of the algorithm in these cases remains reasonably good. Although the scenarios presented in this paper were non-trivial, the future work adresses two principal aspects: 1) to extend these scenarios to include other types of landscape morphologies and 2) to study the other change types provided by the DF1 generator. Additionally a mechanism for the automatic detection of changes must be developed.

Acknowledgments

The authors thank the reviewers for their valuable comments which greatly helped them to improve the contents of this paper.

The first author acknowledges the continuous support received from the Universidad Nacional de San Luis and the ANPCYT. The second author acknowledges support from CONACyT project number 42435-Y.

REFERENCES

Angeline, P. J., Tracking Extrema in Dynamic Environments. In: *Evolutionary Programming VI, 6th International Conference EP'97*, P. J. Angeline, R. G. Reynolds, J. McDonell and R. Eberhart (Eds), pp. 335–345, Lecture Notes in Computer Science Vol. 1213, 1997 (Indianapolis, IN: Springer-Verlag).

Bäck, T., Fogel, D. B. and Michalewicz, Z., (Eds), *Handbook of Evolutionary Computation*, 1997 (Bristol, UK: Oxford University Press).

Blackwell, T., Swarms in Dynamic Environments. In: *Genetic and Evolutionary Computation—GECCO 2003. Proceedings, Part I*, E. Cantú-Paz et al., (Eds), pp. 1–12, Lecture Notes in Computer Science Vol. 2723, 2003 (Chicago, IL: Springer-Verlag).

Blackwell, T. and Branke, J., Multi-swarm Optimization in Dynamic Environments. In: *Applications of Evolutionary Computing: Evoworkshops 2004*,

G. Raidl et al., (Eds), pp. 489–500, Lecture Notes in Computer Science Vol. 3005, 2004 (Coimbra, Portugal: Springer-Verlag).

Branke, J., *Evolutionary Optimization in Dynamic Environments*, 2002 (Boston/Dordecht/London: Kluwer Academic Publishers).

Carlisle, A., Applying The Particle Swarm Optimization to Non-Stationary Environments, PhD thesis, Auburn University, Alabama, USA, 2002.

Carlisle, A. and Dozier, G., Adapting Particle Swarm Optimization to Dynamic Environments. In: *International Conference on Artificial Intelligence*, pp. 429–434, Las Vegas, Nevada, USA, 2000.

Carlisle, A. and Dozier, G., Tracking Changing Extrema with Adaptive Particle Swarm Optimizer. In: *Proceedings of the 5th Biannual World Automation Congress*, pp. 265–270, Orlando, Florida, USA, 2002.

Caro, G. and Dorigo, M., AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 1998, **9**, 317–365.

Esquivel, S. C. and Coello Coello, C. A., On the Use of Particle Swarm Optimization with Multimodal Functions. In: *Proceedings of 2003 Congress on Evolutionary Computation (CEC'2003)*, Vol. 2, pp. 1130–1136, 2003 (Piscataway, NJ: IEEE Press).

Esquivel, S. C. and Coello Coello, C. A., Particle Swarm Optimization in Non-Stationary Environments. In: *Advances in Artificial Intelligence - IBERAMIA 2004*, C. Lemaître, C. A. Reyes and J. A. González (Eds), pp. 757–766, Lecture Notes in Artificial Intelligence Vol. 3315, 2004 (Puebla, México: Springer-Verlag).

Fan, S. S., Liang, Y. and Zahara, E., Hybrid simplex search and particle swarm optimization for the global optimization of multimodal functions. *Engineering Optimization*, August 2004, **36**(4), 401–418.

Guntsch, M. and Middendorf, M., Pheromone Modification Strategies for Ant Algorithms applied to Dynamic TSP. In: *Applications of Evolutionary Computing: Evoworkshops 2001*, E. J. Boers et al. (Eds), pp. 213–222, Lecture Notes in Computer Science Vol. 2037, 2001 (Como, Italy: Springer-Verlag).

Guntsch, M. and Middendorf, M., Applying Population Based ACO to Dynamic Optimization Problems. In: *Ant Algorithms. Third International Workshop, ANTS 2002*, M. Dorigo, G. Di Caro and M. Sampels (Eds), pp. 111–122, Lecture Notes in Computer Science Vol. 2463, 2002 (Brussels, Belgium: Springer-Verlag).

Guntsch, M. and Middendorf, M., A Population based Approach for ACO. In: *Applications of Evolutionary Computing: Evoworkshops 2002*, S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf and G. Raidl (Eds), pp. 72–81, Lecture Notes in Computer Science Vol. 2279, 2002 (Kinsale, Ireland : Springer-Verlag).

Guntsch, M., Middendorf, M. and Schmeck, H., An Ant Colony Optimization

Approach to Dynamic TSP. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, L. Spector et al. (Eds), pp. 860–867, 2001 (San Francisco, CA: Morgan Kaufmann Publishers).

He, S., Prempain, E. and Wu, Q., An improved particle swarm optimizer for mechanical design optimization problems, *Engineering Optimization*, October 2004, **36**(5), 585–605.

Hu, X. and Eberhart, R. C., Adaptive Particle Swarm Optimization: Detection and Response to Dynamic Systems. In: *Proceedings of the 2002 Congress on Evolutionary Computation, CEC'2002*, Vol. 2, pp. 1666–1670, 2002 (Piscataway, NJ: IEEE Service Center).

Janson, S. and Middendorf, M., A Hierarchical Particle Swarm Optimizer for Dynamic Optimization Problems. In: *Applications of Evolutionary Computing: Evoworkshops 2004*, G. Raidl et al. (Eds), pp. 513–524, Lecture Notes in Computer Science Vol. 3005, 2004 (Coimbra, Portugal: Springer-Verlag).

Kennedy, J. and Eberhart, R. C., Particle Swarm Optimization. In: *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995 (Piscataway, NJ: IEEE Service Center).

Kennedy, J. and Eberhart, R. C., *Swarm Intelligence*, 2001 (San Mateo, CA: Morgan Kaufmann Publishers).

Kennedy, J. and Mendes, R., Population Structure and Particle Swarm Performance. In: *Proceedings of the 2002 Congress on Evolutionary Computation, CEC'2002*, Vol. 2, pp. 1671–1676, 2002 (Piscataway, NJ: IEEE Service Center).

Michalewicz, Z. and Fogel, D. B., *How to Solve It: Modern Heuristics*, 2000 (Berlin, Germany: Springer).

Morrison, R. W., Designing Evolutionary Algorithms for Dynamic Environments, PhD thesis, George Mason University, Fairfax, Virginia, USA, 2002.

Morrison, R. W. and De Jong, K. A., A Test Problem Generator for Non-Stationary Environments. In: *Conference on Evolutionary Computation (CEC'99)*, Vol. 3, pp. 2047–2053, (Piscataway, NJ: IEEE Service Center).

Parrot, D. and Li, X., A Particle Swarm Model for Tracking Multiple Peaks in Dynamic Environments using Speciation. In: *2004 Conference on Evolutionary Computation (CEC'2004)*, pp. 98–103, 2004 (Piscataway, NJ: IEEE Service Center).

Ray, T. and Liew, K., A Swarm Metaphor for Multiobjective Design Optimization, *Engineering Optimization*, 2002, **34**(2), 141–153.

Schoonderwoerd, J. B. R., Holland, O. and Rothkrantz, L., Ant-based Load Balancing in Telecommunications Networks, *Adaptive Behavior*, 1996, **5**, 168–207.

Shi, Y. and Eberhart, R., A Modified Particle Swarm Optimizer. In: *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pp. 69–73, 1998 (Piscataway, NJ: IEEE Service Center).

van den Bergh, F., An Analysis of Particle Swarm Optimization. PhD thesis, Faculty of Natural and Agricultural Science, University of Petroria, Pretoria, South Africa, 2002.